MyMusic

Introdução:

Uma cliente possui uma aplicação Java Spring Boot para controle de músicas favoritas de seus usuários. Esta é uma aplicação legada onde o usuário pode interagir com sua lista de músicas favoritas, além disso ele possui um banco de dados que armazena estas informações e um serviço de APIs que realizam a integração deste cliente com o banco de dados.

Após diversas análises de desempenho foi identificado que existe um gargalo na aplicação legada, principalmente na camada de APIs sendo que a mesma está em um servidor monolítico que não comporta a demanda atual e também não permite o fácil e rápido dimensionamento de servidores para atender à demanda.

Foi então solicitado que vocês desenvolvam novos serviços para substituir a camada de APIs utilizando o banco de dados existente, e esta aplicação deve seguir alguns conceitos básicos.

Requisitos:

- **Será construída utilizando técnicas de micro-serviços**, permitindo um dimensionamento independente de APIs, agrupadas de acordo com os domínios de negócios identificados abaixo.
    - Planeje como o deploy da aplicação será feito (12factor);
- **Ser documentada**, Possuir página de documentação de APIs utilizando algum framework consolidado. Utilize boas práticas de APIs REST e organização/nomenclatura do código
- **Manter compatibilidade legado,** utilizar o banco de dados disponibilizado que contém todas as informações do legado.
- **Qualidade de código**, Pensar numa maneira de garantir a qualidade do código sendo que a intenção é evoluir as rotas existentes com certa frequência.

Narrativa de Negócio:

As funcionalidades básicas da aplicação que acessam as APIs compreendem em 5 ações principais descritas abaixo

- Permitir o usuário buscar novas músicas no banco de dados:
    1. O serviço deve validar se usuário informou ao menos 2 caracteres, retornando um HTTP 400 caso a consulta tenha menos de 2 caracteres.
    2. A busca deve ser realizada tanto nas colunas de nome do artista e nome da música.
    3. A busca por música não deve ser case sensitive.
    4. A busca deve retornar valores contendo o filtro, não necessitando de ser informado o nome completo da música ou artista.
    5. O retorno deve estar ordenado pelo nome do artista e depois pelo nome da música.
- Permitir o usuário escolher as músicas do resultado da busca que deseja adicionar na sua playlist:
    1. Deve receber um request contendo o identificador da música e o identificador da playlist.
    2. Deve validar se o identificador da música e o identificador da playlist existem.
- Permitir o usuário remover músicas de sua playlist:
    1. Deve receber um request contendo o identificador da música e o identificador da playlist.
    2. Deve validar se o identificador da música e o identificador da playlist existem.

Narrativa Técnica:

Multi-plataforma:

- Deverá ser utilizado JAVA 11, Spring Boot e Maven. Recomendamos a IDE intelliJ.

Versionamento:

- Cada grupo deverá atraves do comando fork no repositorio original, decidir a estrategia de versionamento e apresentar no fim da sprint 0.

Micro-serviços:

- Existem dois domínios de negócios que foram identificados e suas APIs devem ser construídas permitindo a possibilidade de serem “levantadas” em servidores de forma independente.
    1. Músicas - Busca de Músicas: Este domínio deve conter API de busca de Músicas
    2. Playlist - Controle de Playlist: Este domínio deve conter APIs de busca e alterações de playlist.

Autenticação e Autorização:

- Login (Autorização e Autenticação)
- Logout
  - Retorno estatico
  - Limpeza de cache
- Conta Premium (Conteúdos exclusivos)
  - Rota para acessos de perfis exclusivos

Usuario:

- API de Playlist de Usuário
  - Rota: /api/playlists?user='Robson'
  - Parâmetros: QueryString: {user} string - Obrigatorio
  - Retorno: Objeto "Playlist" do modelo Json
  - Erros tratados: 204 quando não encontrar usuário

- Dados de Conta
  - Cadastro Usuario
  - Tipo de usuario

Musicas:

- API de listagem de Músicas:
  - Rota: /api/musicas?filtro='Bruno+Mars'
  - Parâmetros: QueryString: {filtro} string - Opcional
  - Retorno: Array do objeto "Musica" do modelo Json
  - Erros tratados: 204 com array vazio quando não houver dados e 400 quando caracteres de busca menor que 3

Playlist:

- API de Adicionar relação de Músicas na Playlist
  - Rota: /api/playlists/{playlistId}/musicas
  - Parâmetros:
    - Url: Path param: {playlistId} string
    - Body: Array do objeto "Musica" do modelo Json
  - Retorno: 200 OK
  - Erros tratados: 400 quando identificadores não forem encontrados no banco.

- API de Remover relação de Músicas da Playlist
  - Rota: /api/playlists/{playlistId}/musicas/{musicaId}
  - Parâmetros:
    - Url: {playlistId}
  - Retorno: 200 OK
  - Erros: 400 quando identificadores não forem encontrados no banco.

Banco de dados:

- O banco de dados é um SQLite e está localizado no diretório: "./MyMusic.db"

Documentação:

- As APIs devem estar documentadas na home de cada API..

Desempenho:

- Na API de GET de músicas, pensar numa maneira de guardar pesquisas pelo mesmo termo por 10 min.

Modelo de Dados:

JSON:

Objeto Musica

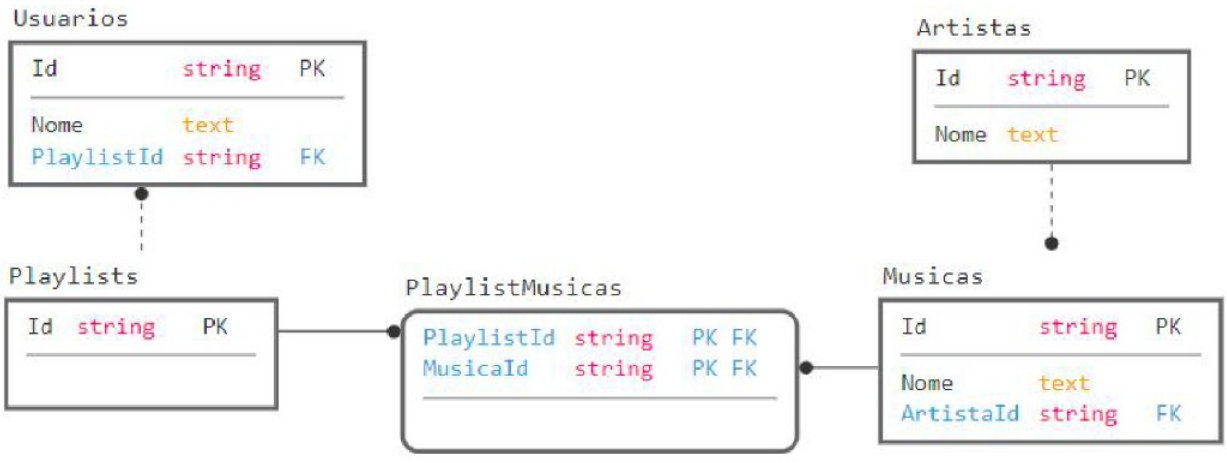
```
[
  {
    "artista": {
      "id": "string",
      "nome": "string"
    },
```

```
      "artistaId": "string",
      "id": "string",
      "nome": "string"
    }
]
```


Objeto Playlist


```
{
  "id": "string",
  "playlistMusicas": [
    {
      "musica": {
        "artista": {
          "id": "string",
          "nome": "string"
        },
        "artistaId": "string",
        "id": "string",
        "nome": "string"
      },
      "musicaId": "string",
      "playlistId": "string"
    }
  ],
  "usuario": {
    "id": "string",
    "nome": "string",
    "playlistId": "string"
  }
}
```


Banco:



Atenção: Os campos Id que utilizam GUID mapear como string devido à complexidade na compatibilidade com o UUID nativo do Java.

Dica:

- Não é necessário, porém é possível utilizar uma ferramenta para abrir e visualizar o arquivo MyMusic.db, de maneira mais fácil, como: https://sqlitestudio.pl/index.rvt